ESC-TR-2003-072

Project Report
PCA-IRT-4

Preliminary Design Review: Kinematic Tracking for the PCA Integrated Radar-Tracker Application

W.G. Coate

25 February 2003
Issued 6 February 2004

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

Prepared for the Defense Advanced Research Projects Agency
under Air Force Contract F19628-00-C-0002.

This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by DARPA/ITO under Air Force Contract F19628-00-C-0002.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Gary Tutungian
Administrative Contracting Officer
Plans and Programs Directorate
Contracted Support Management

Massachusetts Institue of Technology
Lincoln Laboratory

Preliminary Design Review: Kinematic Tracking for the PCA Integrated Radar-Tracker Application

W.G. Coate
Group 102

Project Report PCA-IRT-4

25 February 2003

Issued 6 February 2004

Lexington Massachusetts

ABSTRACT

A kinematic tracker is a classical tracking system. The idea behind kinematic tracking is to store positions and velocities and then, using this information, associate incoming targets with the stored profiles. Using these new targets, the tracker updates its information about positions and velocities to prepare to do the entire process again when it is given the next set of targets.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

A kinematic tracker is a classical tracking system. The idea behind kinematic tracking is to store positions and velocities and then, using this information, associate incoming targets with the stored profiles. Using these new targets, the tracker updates its information about positions and velocities to prepare to do the entire process again when it is given the next set of targets.

This report discusses kinematic tracking as it was adapted for use in Feature-Aided Tracking (FAT [5]), which will be used as the tracker component for a later release of the Integrated Radar-Tracker Application for the PCA program (initially only this kinematic tracker will be used). An overview of this entire application can be found in [3] and discussion of the Ground Moving Target Indicator (GMTI) radar component being used is contained in [7]. This report was mainly derived from a tracker implementation and its documentation by L. Keith Sisterson, but includes modifications made by Duy Nguyen for the FAT system as well.

1.1 KINEMATIC TRACKING—HIGH LEVEL DESCRIPTION

A tracker will receive information from the GMTI system regarding new target detections. On the radar side this is often referred to as the detection list. FAT and the kinematic tracker it was built upon refer to these either as measurements or target reports. These target reports make up half of the input to a tracker. The other half of the input comes from the tracker itself, these being the tracks which were its output from the previous scan. The track histories are referred to, and drawn, as a loop-back input as opposed to persistent data; this reference is done for two reasons. The first reason is that radar is a streaming application, and the output comes in discrete bursts (each burst referred to as a scan). Trackers then are often thought of similarly because they are part of an overall radar system (even though they are arguably closer to thread-based than stream-based computation). The second reason is that the track history for scan n is the result from scan $n-1$, and drawing it as a loop-back system highlights this. A common reference used in the remainder of this report will be a "track/report pair." Pairing a track and a report means that the report is considered to be the next part of the track. A tracker must, in some manner, consider all possible track/report pairs to determine the continuation of the tracks.

1.1.1 Kinematic Tracking Overview

The diagram in Figure 1 is a convenient way to conceptualize the basic tasks which a kinematic tracker must perform. Figure 1 diverges from the actual algorithm in that the stages of "Hypothesize Associations," "Extrapolate Tracks," and "Compute Kinematic χ^2" cannot so conveniently be separated in a serial fashion. In the process of hypothesizing associations, one must extrapolate tracks along different movement models and compute a kinematic χ^2 value for each. In this report, four movement models are discussed; greater or fewer could be used as is appropriate for the specific application. The meanings of these three steps will be explained separately, but the algorithmic implementations are too closely intertwined to separate and will be treated together.

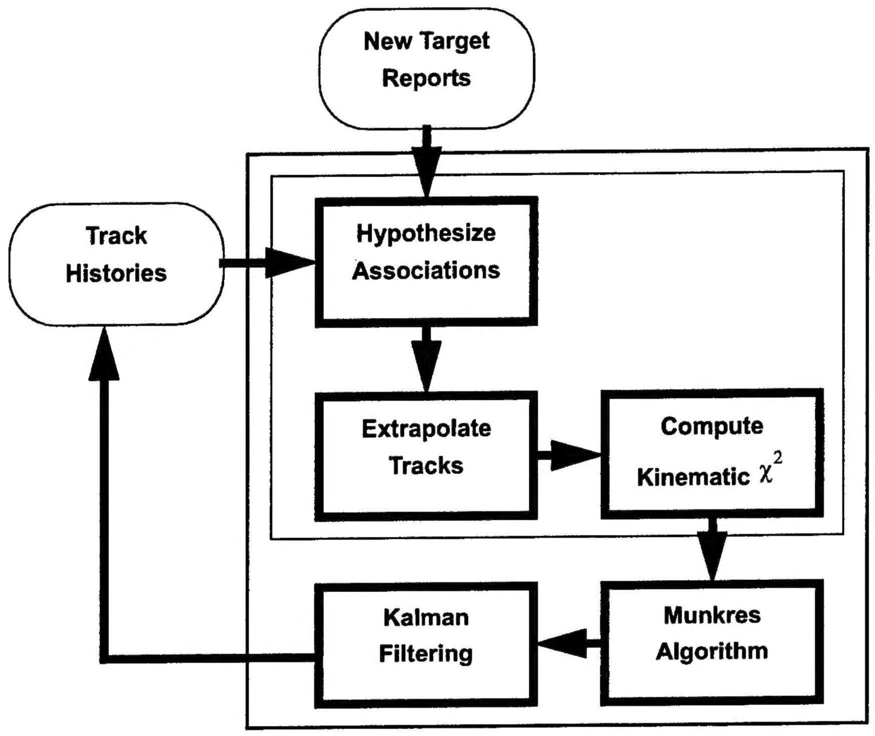

Figure 1. Kinematic tracker logical diagram.

Hypothesizing associations considers all track/report pairs and determines which have a possibility of being correct associations. First, it will be considered if the pairing is possible. Then all possible pairings will need to be extrapolated along some number (three for this example) of known movement hypotheses and an unmodeled maneuver hypothesis. Then these extrapolations must be judged to determine which, if any, gives the most likely manner in which the specific target report could be the continuation of the specific track. The metric used to determine this (also the overall likelihood that a track/report pair is a correct pairing) is the kinematic χ^2 value (where smaller χ^2 values indicate greater likelihood). The Munkres algorithm is a method for finding the optimal assignment of tracks to reports given a matrix of their χ^2 values. A Kalman filter is then used to update the tracks to take their newly associated target reports into account.

A track itself is simply a manner of keeping track of an object across scans. Objects may stop and begin moving again or move in and out of the area of interest, so the notion of beginnings and endings of tracks needs to be dealt with (note that this may not be the case in the initially distributed data set, but false detections and indistinguishable targets will still make this necessary). To deal with a track beginning, tracks are separated into three different kinds of tracks: New, Novice, and Established. A New track is a track that is just starting and has only the initial target report associated with it. Upon receiving another associated target report, a New track will become a Novice track. A track is considered a Novice track as long as any Doppler ambiguity exists. (Doppler ambiguity will be discussed in the next paragraph) Once the Doppler ambiguity of a track has been eliminated, it becomes an Established track. Tracks are removed when a certain period of time or number of scans has passed with no further associations to update them. This would logically take place as part of, immediately before, or immediately after the Kalman filtering stage.

The phenomenon of Doppler ambiguity rises because the measurements of Doppler frequency which the MTI stage makes are modulo the pulse repetition frequency (PRF). This means that two targets that differ in Doppler frequency by an integer multiple of the PRF will be seen as having the same Doppler frequency. Note that if a target has a Doppler frequency equal to an integer multiple of the PRF, it may have already been filtered out by the MTI stage as a stationary object. In *Introduction to Airborne Radar* (Chap. 25, [8]) low PRFs are given to be 250–4000 Hz, medium PRFs to be 10–20 kHz, and high PRFs 100–300 kHz. The choice of PRF relies on many factors, but generally a Ground Moving Target Indicator (GMTI) radar (which is the MTI portion of this integrated radar-tracker application) will use a low or medium PRF, and is therefore subject to Doppler ambiguity issues. The tracker needs to know the Doppler frequency to calculate the velocity of a target, so it needs to know with greater precision than modulo the PRF. There are various ways to deal with Doppler ambiguities; however, they all require state information, so this will be the job of the tracker in this implementation. For more information on Doppler ambiguities, see section "Potential Doppler Ambiguities" on page 284 of *Introduction to Airborne Radar* [8]. (The case of "PRF Less Than Spread of Doppler Frequencies" applies to this discussion.)

1.1.2 Hypothesize Association, Extrapolate Tracks, Compute Kinematic χ^2

While these three logical operations cannot be readily separated algorithmically, it is algorithmically possible to separate them into two stages. The first stage looks at all track/report pairs and uses geographic location to eliminate impossibilities. The next stage examines each remaining track/report pair individually to first eliminate kinematic impossibilities and then determine the correct motion model and χ^2 value for possible pairings.

1.1.2.1 Reduce By Geographic Position

A sizeable amount of computation is done on each track/report pair which is being considered by the rest of the algorithm. The purpose of this step is to eliminate geographically impossible pairings. Ground vehicles can be assumed to have a low enough velocity that they will not move too great a distance between scans. Using the report position, track position, and postulated maximum ground vehicle speed, only certain track/report pairs will be possible, so all others can be ignored. This implementation will geographically check all tracks for each incoming target report.

1.1.2.2 Examine One Track/Report Pair

For each target report, each track/report pair considered geographically possible by the previous step will be tested. First an unmodeled maneuver model will be extrapolated to determine if the track/report pair is kinematically possible, and if possible, then a χ^2-like value will be calculated (for all track/report pairs deemed geographically or kinematically impossible, the χ^2 value assigned is an arbitrary value large enough to prevent the Munkres algorithm from choosing it over a viable association and no other hypotheses are tested). Now the method of dealing with each pair diverges depending upon the status of the track. For New or Novice tracks, Doppler ambiguity is checked, and the χ^2-like value from the unmodeled test is saved for this pair. For Established tracks, the track is extrapolated in three manners: constant velocity; linear acceleration; and constant speed, arc of circle. Each hypothesis for track extrapolation will be considered in the order above and yield a χ^2 value. Each will also have a separate limit for consideration and use this to determine success or failure. If a motion model succeeds, the model name and χ^2 are saved. In the case of failure, the next motion model is checked. After all models are exhausted, the χ^2-like value from the unmodeled test will be stored and "unmodeled" will be stored as the motion model name. If the scan times are close together in time, most motion will be seen as constant velocity. Note that the number and type of hypothesized motion models is dependent upon the application, and more or less could be used.

1.1.3 Munkres Algorithm

The Munkres algorithm [4] is a method of uniquely associating things, one with another, while minimizing a cost function. The cost function is used to populate a matrix of all possibilities. The Munkres algorithm then chooses associations by choosing exactly one member of each row and column such that the sum of the chosen elements is minimized. In a kinematic tracker, the Munkres algorithm is one option to use to decide which reports will be assigned as the successors of which tracks. The original algorithm requires a square input matrix and will associate all items. An extended version of this algorithm [1] is used here as it will work on non-square matrices and associates as many items as the minimum of rows and columns. The matrix passed in is made of the kinematic χ^2 (or merged χ^2 for FAT) values. Recall that, for track/report pairs determined to be geographically or kinematically impossible, a large χ^2 value will have been used to prevent their selection if any other possible choices are available. The Munkres algorithm will perform all optimal pairings, then will continue pairing those intended as impossible afterward. Additional checks have been added to be sure no impossible pairings are passed to the Kalman filter as being associated.

1.1.4 Kalman Filtering

A Kalman filter is a classical way to update knowledge with measurements. The idea behind a Kalman filter is that any measurement has an associated uncertainty. Different measurements may have different uncertainties. By using multiple measurements and taking their respective uncertainties into account, a more accurate estimate may be made. It would be impractical to store each and every measurement, especially considering that the total number of measurements may be neither known nor likely to be few. The Kalman filter deals with this by taking two separate measurements with two separate errors. Using this information, it determines a new estimate based on both measurements, which will presumably be a better estimate than either measurement separately. The Kalman filter will also determine an error for the new estimate. In this way, only one former state need be saved, and each new measurement is simply applied to the aggregate history and a new aggregate history is formed. In the kinematic tracker, a Kalman filter is used to update the track histories with their new target reports. An intuitive explanation of Kalman filtering can be found in [6], and a more rigorous description can be found in [2]. (Section 5.5 of [2], "The discrete Kalman filter," starting on p. 214 discusses a Kalman filter similar to the one used in this tracker in greater depth than covered here.)

1.2 PARAMETERS

This section explains parameters which are needed as inputs and controls for the operation of the kinematic tracker. The output values are also discussed. Because the tracker works as a loop-back system, it will be taken as understood that the output for one stage is also the input of the next, and outputs will not be listed twice below.

1.2.1 Input Parameters

There is really only one external input, the target-report list. The target-report list will be signified by the one-dimensional array `M[]`, each element of which will be a structure. `M[]` will have a member structure named `cent` (short for centroid). The target-report list will be what will populate `cent` in the appropriate member of `M[]`. Table 1 outlines the data members of the elements of `cent` which need to be populated before passing them to the tracker. Shaded elements are data members used by later stages of the overall FAT algorithm, but not by kinematic tracking. Note that `cent` was based upon the input structure used in the system this tracker was based upon, and a different input structure is used by the GMTI system attached here.

TABLE 1

Pre-Populated Data Members of `cent`

Data Member Name	Explanation
`rg`	Range position of target. This is used primarily to determine ground position.
`az`	Azimuth position of target. This is used primarily to determine ground position.
`dop`	Doppler measurement for target. This is used primarily to measure the target's radial velocity.
`time`	Time stamp for when this detection was made.
`snr`	Signal-to-noise ratio (SNR) of detection. This is a measure of the relative strength of the target detection.
`hrr`	High range resolution (HRR) profile for target.

1.2.2 Output Parameters

The output is also composed of a structure. It is the list of tracks recognized by the kinematic tracker. The track list will be signified by the one-dimensional array `T[]`. The purpose of the track list is two-fold. One is as output to a user; the other is as loop-back input to the tracker. Table 2 outlines the data members of the elements of `T[]` which are used outside the current scan with the kinematic tracker. Shaded elements are data members not of direct interest to a human user, but are used either in the loop-back input or in other sections of the overall FAT algorithm.

TABLE 2

Pre-Populated Data Members of `T[]`

Data Member Name	Explanation
`snr`	SNR of last target report associated with this track. This is a measure of the relative strength of that target detection.
`x`	Estimated x-coordinate of target at time `time`.
`y`	Estimated y-coordinate of target at time `time`.
`x_dot`	Estimated velocity of target in the x direction at time `time`.
`y_dot`	Estimated velocity of target in the y direction at time `time`.
`time`	Time stamp for the last target report associated with this track.
`status`	{New, Novice, Established}
`class`	Classification vector from overall FAT computations.
`aspect`	Estimated aspect angle of target at time `time`.
`hrr`	High range resolution (HRR) profile for last target report associated with this track.
`Pp`	Extrapolated process noise covariance matrix.
`Hypothesis`	Movement model hypothesized for last track/report association for this track.

1.2.3 Control Parameters

There are multiple control parameters. In the implementation these will be part of a structure (see Section 3.4). However, a structure is not necessary, so for the algorithm they will be treated as separate values. Parameters important to the algorithmic description or general conceptual understanding are listed in Table 3.

TABLE 3

Control Parameters for Kinematic Tracker

Parameter Name	Explanation
MaxNumReports	Maximum number of target reports allowed (necessary only for worst-case analysis or memory restrictions of actual implementation). This parameter is not used in the Matlab implementation.
MaxNumTracks	Maximum number of tracks allowed (necessary only for worst-case analysis or memory restrictions of actual implementation). This parameter is not used in the Matlab implementation.
maxSpeed	Assumed maximum possible speed for a target.
maxAccel	Assumed maximum possible acceleration for a target.
RVar	Variance in range measurement from GMTI stage.
AzVar	Variance in azimuth measurement from GMTI stage.
DopVar	Variance in Doppler measurement from GMTI stage.
CVHLimit	Limit used to test constant velocity hypothesis.
LATHLimit	Limit used to test linear acceleration hypothesis.
ArcHLimit	Limit used to test constant speed, arc of circle hypothesis.
cvsq	Used to compute process noise covariance matrix for constant velocity hypothesis.
latsq	Used to compute process noise covariance matrix for linear acceleration hypothesis.
mnsq	Used to compute process noise covariance matrix for constant speed, arc of circle and unmodeled hypotheses.
NewLimit	Limit to how long a New track is kept if not updated.
NoviceLimit	Limit to how long a Novice track is kept if not updated.
EstablishedLimit	Limit to how long an Established track is kept if not updated.
GridXNearRatio	Ratio to define the concept of a track being "near" a target report in the X direction for geographic decimation.
GridYNearRatio	Ratio to define the concept of a track being "near" a target report in the Y direction for geographic decimation.
PlatformXPos	X position of radar platform by the XY grid used for targets (Y position of platform is 0).

2. FUNCTIONAL OVERVIEW

This section will cover the base functionality required by the kinematic tracker. Only the general required functionality of a kinematic tracker is discussed here. The methods in this section for performing operations are generalizations of what is necessary and required for the operations, and may not be reflections of the Matlab implementation. This is done to hopefully help highlight areas open to new implementations to take advantage of PCA hardwares' unique characteristics. The section "Implementation Considerations" will contain some discussions about basic performance improvements. Functions which act as translators between systems, parameter filters, and other implementation "glue," though necessary, are not discussed in this report.

In this section (and throughout the report) C++-style pseudocode is used, although Matlab code is provided for implementing this tracker. This was chosen mainly to avoid the potential confusion of variable types in Matlab.

2.1 HYPOTHESIZE ASSOCIATION, EXTRAPOLATE TRACKS, COMPUTE KINEMATIC χ^2

These three stages are once again divided along the lines of an initial decimation by geographic position and then an examination of each remaining track/report pair.

2.1.1 Reduce By Geographic Position [`get_tracks_near()`]

Input: This operation requires as input the track list `T[]` and target report list `M[]`. The sizes of these lists will vary from scan to scan; however, they may be bounded by *MaxNumTracks* and *MaxNumReports*.

Output: The output of this operation is logically a boolean association matrix `Assoc`. `Assoc` will be of size $t \times m$, where t is the number of elements in `T[]` and m is the number of elements in `M[]`.

All track/report pairs must be examined. Later stages may perform large computations for each track/report pair that must be operated upon. This stage acts as a first cut. The conceptual algorithm for doing this is as follows:

```
for(int i=0; i<length(M); ++i)
  for(int j=0; j<length(T); ++j)
    if(near(M[i],T[j]))
      Assoc[j][i]=true;
    else
      Assoc[j][i]=false;
```

The function `length()` is considered to return the number of elements of the array which it is passed as input. The function `near()` returns a boolean value indicating if the input target report is sufficiently close to the input track. The actual definition of `near()` is unimportant to the algorithm as long as it always returns true when the track/report pair is a correct association. A function always returning true would be acceptable for the algorithm. Discussions on the concept of being geographically near and on efficiency can be found in the section "Implementation Considerations."

2.1.2 Examine One Track/Report Pair [`associate()`]

Input: This operation will require the track list `T[]`, target report list `M[]`, and boolean association matrix `Assoc`. `Assoc` is of size $t \times m$ where t is the number of elements in `T[]` and m is the number of elements in `M[]`. The number of elements in `T[]` and `M[]` will vary from scan to scan; however, they may be bounded by *MaxNumTracks* and *MaxNumReports*.

Output: The output of this operation will be a possibly modified boolean association matrix `Assoc`, a chosen hypothesis matrix `Hyp`, and a matrix of the calculated kinematic χ^2 values `Kin_X2`. All three of these matrices will be of size $t \times m$ where t is the number of elements in `T[]` and m is the number of elements in `M[]`.

This process will loop over all track/report pairs. If it finds in `Assoc` that the pair has already been marked false, then the equivalent entry in `Kin_X2` will be set to a suitably large value to avoid being considered viable by Munkres (100 chosen for Matlab code). If the pair has been marked true, then an unmodeled test will take place. The unmodeled test computes a limit on the distance between the target report and the extrapolated position of the track. The limit to determine if the unmodeled test is passed corresponds to nine times the variances from all the sources of error plus the distance the target could move (from the initial track position) if it accelerated at *maxAccel*. If the unmodeled test fails, then `Assoc` is modified to mark this pairing as impossible and `Kin_X2` is set to the same large value as it would had it failed previously. If it passes, then a χ^2-like value is computed, made too large to be chosen in place of a modeled association, and saved.

For New or Novice tracks, Unmodeled is saved in `Hyp` and the unmodeled χ^2 is saved in `Kin_X2`. Doppler ambiguity is also examined, but the results do not alter execution of this stage.

For Established tracks, the unmodeled test will have been calculated with slight differences to take more knowledge into account. The primary difference is in the use of the plant-noise matrix `Q`. The plant-noise matrix is dependent upon the difference in time between the target report and last track update and a parameter `q` (full definition of `Q` can be found in Section 2.3). For an Established track, the unmodeled test uses a `Q` with `q` equal to one-third the square of *maxAccel*. After passing the unmodeled test, an Established track tests, in order, the constant velocity hypothesis, the linear accelerating track hypothesis, and the arc-of-circle hypothesis.

2.1.2.1 *Constant Velocity Hypothesis*

The track extrapolation for this hypothesis is the same as the unmodeled computation but with a different plant-noise matrix `Q`. For the constant velocity hypothesis, the parameter *cvsq* is used for `q` in forming `Q`. The main difference is that an actual χ^2 is computed as follows:

$$\chi^2 = \frac{r^2}{\sigma_r^2} + \frac{(d_m - d_t)^2}{\sigma_d^2 + \sigma_{dt}^2} \tag{1}$$

where *d* is a Doppler value, *r* is the radial distance between the extrapolated track and target report, and σ^2 is the variance. The value of σ_r^2 is calculated as follows:

$$\sigma_r^2 = \frac{(x_m - x_t)^2(\sigma_x^2 + \sigma_{xt}^2) + (y_m - y_t)^2(\sigma_y^2 + \sigma_{yt}^2) + 2(x_m - x_t)(y_m - y_t)(\sigma_{xy} + \sigma_{xtyt})}{r^2} \tag{2}$$

The calculated χ^2 is then tested against parameter *CVHLimit*. If it is within the limit, then constant velocity is saved in `Hyp` and the χ^2 is saved in `Kin_X2`. If it is outside the limit, then the next Hypothesis is tested.

2.1.2.2 *Linear Accelerating Track Hypothesis*

The track position and χ^2 are extrapolated as follows:

$$x_t = x_p + \dot{x}t + \frac{\ddot{x}t^2}{2} \tag{3}$$

$$y_t = y_p + \dot{y}t + \frac{\ddot{x}\dot{y}t^2}{2\dot{x}} \tag{4}$$

$$\chi^2 = \frac{\left((\dot{x} + \ddot{x}t)\sin a + \left(\dot{y} + \frac{\ddot{x}\dot{y}t}{x}\right)\cos a - d_m\right)^2}{\sigma_d^2 + \sigma_{dt}^2} + \frac{r^2}{\sigma_r^2} \tag{5}$$

There are two tests for success here. The first is to test that the χ^2 is within the confidence limit parameter *LATLimit*. The second is to make certain that the acceleration computed from a least-squares fit of the target-report position to the extrapolated track position above is not greater than the parameter *maxAccel*. If both of these tests are passed, then linear accelerating track is saved in `Hyp` and the χ^2 is saved in `Kin_X2`. If either test fails, the next hypothesis is tested.

2.1.2.3 Constant Speed Arc-of-Circle Track Hypothesis

The arc of a circle is constructed from the track position and direction compared to the current target report. χ^2 is calculated as follows:

$$\chi^2 = \frac{(\dot{x}_t \sin a + \dot{y}_t \cos a - d_m)^2}{\sigma_d^2 + \sigma_{dt}^2} + \frac{(vt - s)^2}{t^2\sigma_v^2 + \sigma_s^2} \qquad (6)$$

where $v = \sqrt{\dot{x}^2 + \dot{y}^2}$ (speed of the target), s = distance along arc of circle, σ_v^2 = variance of v, and σ_s^2 = variance of s. The only test performed is χ^2 against the parameter *ArcLimit*. If the test is passed, constant speed, arc-of-circle is saved in `Hyp` and the χ^2 is saved in `Kin_X2`.

If all three of these models fail, then Unmodeled is saved in `Hyp` and the previously saved unmodeled χ^2-like value is saved in `Kin_X2`.

2.2 MUNKRES ALGORITHM [`MUNKRES()`]

Input: This general algorithm only requires the matrix of χ^2 values `Kin_X2` (or the merged `FAT_X2` if FAT is being used). To easily include checking for geographically or kinematically impossible associations, the boolean association matrix `Assoc` will also be passed as input. The size of these matrices is $t \times m$, where t is the number of elements in `T[]` and m is the number of elements in `M[]`. The number of elements in `T[]` and `M[]` will vary from scan to scan; however, they may be bounded by *MaxNumTracks* and *MaxNumReports*.

Output: The output is an optimal list `MunkresResult[]` of chosen track/report pairings. The elements of the list will be tuples indicating the pairing. This list will include unassociated tracks and unassociated target reports as being associated with a null value (this implementation used –1 for this null value). The size of the list may be at worst $t + m$ and at best max(t, m), where t is the number of elements in `T[]` and m is the number of elements in `M[]`.

The following is quoted directly from pages 803 and 804 of [1]. In the paper, the original Munkres algorithm is first listed, then changed steps are listed for the extended Munkres algorithm. In the interests of clarity, quotes of the steps from the original algorithm are being substituted in the paper where they belong in the quote of the extended version.

"The statement of the extended algorithm follows.

Preliminaries. (a) $k = \min(n, m)$, no lines are covered, no zeros are starred or primed. (b) *If the number of rows is greater than the number of columns, go at once to step* 0. Consider a particular row of the matrix (a_{ij}); subtract the smallest element from each element in the row; do the same for all other rows. *If the number of columns is greater than the number of rows, go at once to step* 1.

Step 0. (c) For each column of the resulting matrix, subtract from each entry the smallest entry in the column.

Step 1. Find a zero, Z, of the matrix. If there is no starred zero in its row nor its column, star Z. Repeat for each zero of the matrix. Go to step 2.

Step 2. Cover every column containing a 0*. If k columns are covered, the starred zeros form the desired independent set. Otherwise, go to step 3.

Step 3. Choose a noncovered zero and prime it; then consider the row containing it. If there is no starred zero Z in this row, go to step 4. If there is a starred zero Z in this row, cover this row and uncover the column of Z. Repeat until all zeros are covered. Go to step 5.

Step 4. There is a sequence of alternating starred and primed zeros constructed as follows: let Z_0 denote the uncovered 0'. Let Z_1 denote the 0* in Z's column (if any). Let Z_2 denote the 0' in Z_1's row. Continue in a similar way until the sequence stops at a 0', Z_{2k}, which has no 0* in its column. Unstar each starred zero of the sequence, and star each primed zero of the sequence. Erase all primes and uncover every line. Return to step 2.

Step 5. Let h denote the smallest noncovered element of the matrix; it will be positive. Add h to each covered row; then subtract h from each uncovered column. Return to step 3 without altering any asterisks, primes, or covered lines."

Note that the above algorithm terminates through step 2. The result `MunkresResult[]` will be generated by using the coordinates of the zeros of the resultant matrix. The Munkres algorithm may have chosen geographically or kinematically impossible associations where no possible association was available. To eliminate these, each response pair will be tested against `Assoc`. If a pairing is found which is marked impossible within `Assoc`, then the original pair will be replaced by two new pairs, with each element of the first pair paired with some sort of null value (–1 is used in the provided Matlab code). After the entire `MunkresResult[]` list is checked, then the matrix result from the actual Munkres algorithm will be examined for rows or columns containing no zero (which must exist if the matrix is not square). The equivalent tracks or target reports will each be appended to the list in a new tuple with the null value as their partner.

2.3 KALMAN FILTERING [`Kalman()`]

Input: This operation will require as input the optimal pairing list `MunkresResult[]`, the track list `T[]`, and the target report list `M[]`. The size of an element of `MunkresResult[]` is the size of two references (integer index, pointer, or whatever else might be convenient for the given implementation), and the length of the list is at worst $t + m$ and at best max(t, m), where t is the number of elements in `T[]` and m is the number of elements in `M[]`. The number of elements in `T[]` and `M[]` will vary from scan to scan; however, they may be bounded by *MaxNumTracks* and *MaxNumReports*.

Output: The updated track list `T[]` is the output. The length of `T[]` will vary from scan to scan. Recall that an actual Kalman filter is run only on Established tracks, so the Kalman filter itself will not change the size of `T[]`. The additional step of creating new tracks with unassociated target reports will increase the size of `T[]`, and the possible removal of tracks not updated in a certain specified time will decrease the size of `T[]`, so the output and input sizes of `T[]` need not match. The size can still be considered bounded by *MaxNumTracks* though.

For New tracks, covariance matrices are initialized and relevant initial information is stored. For Novice tracks, velocity windows are examined to determine if there is enough information to resolve Doppler ambiguity. If there isn't enough information, the target report is saved. Once there is sufficient information, then the state is set to Established and the Kalman filter is recursively run on all previously saved target reports (in chronological order) and then on the current one (the maximum set in the original code is five saved target reports plus the current one). For Established tracks, an actual Kalman filter will be performed. A description of the operation in matrix notation (along with matrix definitions) is below. Note that the subscript k indicates the current scan, so k–1 indicates the most recent prior scan:

The true current state:

$$x_k = \phi_{k-1} x_{k-1} + noise \tag{7}$$

1) The current target report can then be represented as follows:

$$z_k = H_k x_k + noise \tag{8}$$

(where $H_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \sin a & 0 & \cos a \end{bmatrix}$ with a = azimuth angle)

2) The initially extrapolated next state for the track:

$$x_k^- = \phi_{k-1} x_{k-1}^+ \tag{9}$$

3) The error covariance matrix for the initially extrapolated next state:

$$P_k^- = \phi_{k-1} P_{k-1}^+ \phi_{k-1}^T + Q_{k-1} \tag{10}$$

4) The Kalman gain stage 1:

$$^{1}K_k = P_k^- H_k^T [H_k P_k^- H_k^T + R_k]^{-1} \tag{11}$$

(where $H_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$)

5) The error covariance matrix is then updated:

$$P_k^+ = [I - {}^{1}K_k H_k] P_k^- \tag{12}$$

(where $H_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$)

6) The extrapolated track covariance matrix is set to the updated track covariance matrix:

$$P_k^- = P_k^+ \tag{13}$$

7) The Kalman gain stage 2:

$$^{2}K_k = P_k^- H_k^T [H_k P_k^- H_k^T + \sigma_d^2]^{-1} \tag{14}$$

(where $H_k = \begin{bmatrix} 0 & \sin a & 0 & \cos a \end{bmatrix}$ with a = azimuth angle)

8) The final error covariance update:

$$P_k^+ = [I - {}^{2}K_k H_k] P_k^- \tag{15}$$

(where $H_k = \begin{bmatrix} 0 & \sin a & 0 & \cos a \end{bmatrix}$ with a = azimuth angle)

9) The state estimate update:

$$x_k^+ = x_k^- + K_k[[I - BK_k X][z_k - H_k x_k^-]] \tag{16}$$

(where $H_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \sin a & 0 & \cos a \end{bmatrix}$, $K_k = \begin{bmatrix} ^{1}K_k & ^{2}K_k \end{bmatrix}$, $B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \sin a & 0 & \cos a \end{bmatrix}$, and $X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

with a = azimuth angle)

The meanings or definitions of the other matrices are as follows:

$x_k, x_k^-,$ or x_k^+ : $\begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix}$ where x and y are coordinate positions and $\dot{x}$ and $\dot{y}$ are velocities in the respective coordinate directions. These are members x, x_dot, y, and y_dot, respectively, of elements of T[].

z_k : $\begin{bmatrix} x \\ y \\ d \end{bmatrix}$ where x and y are coordinate positions and d is the velocity in the radial direction. These are members x, y, and dop, respectively, of elements of M[].

$\phi_{k-1} = \begin{bmatrix} 1 & t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t \\ 0 & 0 & 0 & 1 \end{bmatrix}$ where t is the time between the target report and last track update.

P_k^- or P_k^+ : $\begin{bmatrix} P_1 & P_2 & P_3 & P_4 \\ P_2 & P_5 & P_6 & P_7 \\ P_3 & P_6 & P_8 & P_9 \\ P_4 & P_7 & P_9 & P_{10} \end{bmatrix}$ P_k^- is referred to as Pm and P_k^+ as Pp. These are stored as members of elements of T[]. These are covariance matrices for the updated states generated by the Kalman filter.

$Q_{k-1} = \begin{bmatrix} \frac{qt^3}{3} & \frac{qt^2}{2} & 0 & 0 \\ \frac{qt^2}{2} & qt & 0 & 0 \\ 0 & 0 & \frac{qt^3}{3} & \frac{qt^2}{2} \\ 0 & 0 & \frac{qt^2}{2} & qt \end{bmatrix}$ where t is the time between the target report and last track update and q is a constant dependent upon the motion model last chosen to describe track movement. This is stored as member Q of elements of T[].

$R_k : \begin{bmatrix} R_1 & R_2 \\ R_2 & R_3 \end{bmatrix}$ is the target report measurement covariance matrix. It is calculated using the MTI results and the parameters *RVar*, *AzVar*, and *DopVar*, which would be taken from the MTI's system specifications. It is stored as member `R` of elements of `M[]`.

${}^1K_k : \begin{bmatrix} K_1 & K_2 \\ K_3 & K_4 \\ K_5 & K_6 \\ K_7 & K_8 \end{bmatrix}$ is calculated and used within the Kalman filter. It is stored as member `K1` of elements of `T[]`.

${}^2K_k : \begin{bmatrix} K_9 \\ K_{10} \\ K_{11} \\ K_{12} \end{bmatrix}$ is calculated and used within the Kalman filter. It is stored as member `K2` of elements of `T[]`.

I: identity matrix of appropriate size for computation.

After this computation, x_k^+ (the estimated position for the track) is the nominal result, and x_k^+ and P_k^+ are stored for use in the next iteration. The above steps 1–10 describe a Kalman filter. Other computations listed in this section are related in purpose for the kinematic tracker, but are not actually part of a Kalman filter. The Kalman filter portion may be extracted for reuse in FAT for the intermediate deliverable. All track/report pairs for FAT will be extrapolated with a Kalman filter and their aspect angle will then be estimated (discussed in Appendix A).

It should be noted that only elements `x`, `x_dot`, y, `y_dot`, `status`, and `Pp` of elements of `T[]` are updated by the above operations. To fully update a track, the other required elements (`snr`, `time`, `class`, `aspect`, `hrr`, and `Hypothesis`) are directly replaced by the value belonging to the target report that the track was just associated with.

The final step which needs to be performed to update the track list is to examine each track in `MunkresResult[]`, which is paired with a null value. The `time` member of the track will be compared against the `time` member on target reports in the current scan. If this difference is larger than that allowed in the limit for tracks of its status (parameters *NewLimit*, *NoviceLimit*, and *EstablishedLimit*), then the track is removed. If *MaxNumTracks* is exceeded, then tracks may be removed by a chosen policy. One policy might be to decrease the time allowed for waiting on associations before being removed. Another might be to remove the tracks with the lowest `snr`. Note that the Matlab code delivered does not actually use a *MaxNumTracks* value, so no tracks will be dropped by the Matlab code for this reason.

3. DATA STRUCTURES

A variety of data structures are used in the tracker. Below is a discussion on the initial proposition for their specific structure. The definitions will be given in a C style (note that in C pointers are generally used to declare dynamically resizable arrays).

3.1 TARGET REPORTS

The structure for target reports is a simple structure, the global list of which has been referred to as `M[]` in this report. The centroid structure will contain the information from the MTI report which will be saved in the target report. Additional information (such as the cosine of the azimuth angle) is stored in this structure for convenience.

```
struct target_report
{
 struct centroid cent; //contains original info passed in
                       // from MTI
 double Range;         //These three parameters are the
 double Azimuth;       // ground values of rg, az, & dop
 double Doppler;       // in cent (the values in cent may
                       // be absolute or may be indices).
 double sin_az;        //sine of the azimuth angle
 double cos_az;        //cosine of the azimuth angle
 double sdsqd;         //doppler variance (in m/s)
 double abs_dop;       //corrected doppler (in m/s)
 double R[2][2];       //measurement (x,y) covariance
                       // matrix
}
```

```
struct centroid
{
 int rg;                //range gate index or absolute
                        // range in meters for this target
 int az;                //beam index or absolute azimuth
                        // angle in radians for this target
 int dop;               //doppler bin index or absolute
                        // doppler in m/s for this target
 double time;           //time stamp associated with this
                        // target
 double snr;            //SNR for this target
//double hrr[hrr_size];
   //will be introduced for FAT.
                        //vector containing the HRR
                        // profile for this target
   //additional members to be populated by the tracker
   // all this information should be redundant with the
   // above info.
 double x;              //x coordinate of this target
 double y;              //y coordinate of this target
}
```

3.2 TRACK

The track structure is a simple structure, the global list of which has been referred to in this report as `T[]`. This structure is used both as output and input, though only a subset of the structure is required in the following scan.

```
struct track_history
{
 char status[];                      //string containing status
                                     // (enum should be used in
                                     // compiled
                                     // implementation)
 double x_pos;                       //x coordinate of
                                     // estimated position of
                                     // track
 double y_pos;                       //y coordinate of
                                     // estimated position of
                                     // track
 double x_vel;                       //estimated x direction
                                     // velocity of track
 double y_vel;                       //estimated y direction
                                     // velocity of track
 double snr;                         //snr from last associated
                                     // target
 double time;                        //time from last target
// double hrr[hrr_size];
   //will be introduced for FAT.

                                     //hrr from last target
// struct classification class;
   //will be introduced for FAT.

                                     //prior classification
                                     // vector, computed by
                                     // Bayesian classifier.

// double aspect;
   //will be introduced for FAT.

                                     //estimated aspect angle
                                     // of track in radians
   //additional members
 struct VelWindow vw[5];             //represents different
                                     // possible actual
                                     // doppler values, used in
                                     // resolving doppler
                                     // ambiguity.
 struct CVHFilter cvhf;         //Abstraction from
                                // original system, may be
                                // removed.
 struct target_report Msave[5]; //saves old targets for
                                // New/Novice tracks until
                                // they become established
 char Hypothesis[];             //string containing
                                // hypothesis (enum should
                                // be used in compiled
                                // implementation)
 double Q[4][4];                //process noise covariance
                                // matrix
 double Pm[4][4];               //initial extrapolation
                                // covariance matrix
 double Pp[4][4];               //updated extrapolation
                                // covariance matrix
 double K1[4][2];               //Kalman gain stage 1
                                // matrix
 double K2[4][1];               //Kalman gain stage 2
                                // matrix
}
struct VelWindow
{
 int active;                    //indicates state of this
                                // velocity window
 double xdot;                   //x direction velocity
 double ydot;                   //y direction velocity
 double xdotvar;                //variance in x direction
                                // velocity
 double ydotvar;                //variance in y direction
                                // velocity
 double xydotvar;               //covariance between x and
                                // y velocities
}
struct CVHFilter
{
 double x;                      //x coordinate
 double y;                      //y coordinate
 double xdot;                   //x velocity
 double ydot;                   //y velocity
}
```

3.3 ASSOCIATION MATRICES

These matrices are simply $t \times m$ matrices, where t is the length of the global track list for the current scan and m is the length of the global target report list for the current scan. Because the sizes of these lists may change from scan to scan, the dimensions of these matrices also may change. There will be three or five matrices of this size. For simple kinematic tracking, the matrices are `Assoc`, `Hyp`, and `Kin_X2`. For FAT, the additional matrices `CAT_or_SAT_X2` and `FAT_X2` are also used (though `Kin_X2` will be copied to `FAT_X2` in simple kinematic tracking). A dynamically resizable matrix in C is represented by a double pointer. Because a string in C is usually represented by a `char*`, a matrix of strings will be represented as a triple pointer. A matrix of arrays would also be represented as a triple pointer.

```
bool **Assoc;                //T/F matrix based on geographic
                             // feasibility
char* **Hyp;                 //matrix of strings keeping track
                             // of which hypothesis was last
                             // chosen (enum should be used in
                             // compiled implementation)
double **Kin_X2;             //contains X^2 values for
                             // kinematic tracker
double **CAT_or_SAT_X2;      //contains X^2 values for CAT or
                             // SAT (whichever is used)
double **FAT_X2;             //contains final/merged X^2
                             // values.
```

3.4 TRACKER PARAMETERS

This simple structure holds the control parameters used for the tracker. These will be set by an initialization function and perhaps never changed after that.

```
struct TrackerParameters
{
 bool UsingExact;           //specifies if centriods report
                            // absolute or index values.
 double SpeedOfLight;       //Make sure all parts of IRT
                            // use same approximation.
 long MaxNumReports;        //maximum number of reports
                            // considered
 long MaxNumTracks;         //maximum number of tracks kept
     //the following four parameters are used to determine
     // how far a target may move between scans.
 double maxSpeed;           //maximum speed considered
                            // (m/s)
 double maxSpeedsqd;        //maxSpeed*maxSpeed
 double maxAccel;           //maximum acceleration
                            // considered (m/s^2)
 double maxAccelsqd;        //maxAccel*maxAccel
     //the following three parameters are used to
     // formulate measurement covariance matrix R
 double InvalidChiSqd;      //Chi squared value to indicate
                            // impossible pairing
 double MaxPosDiff;         //Maximum possible difference
                            // between target report and
                            // unmodeled extrapolation
 int NumAzBins;             //The number of azimuth bins
                            // used by GMTI
 double TotalAzCoverage;    //Total azimuth coverage of
                            // GMTI in radians.
 double AzDegPerBin;        //The degree extent of one
                            // azimuth bin
 long NumRangeBins;         //The number of range bins used
                            // by GMTI
 int NumDopBins;            //The number of doppler bins
                            // used by GMTI
 double DopMpSPerBin;       //The coverage (in m/s) of one
                            // doppler bin.
```

```
    //the following three parameters represent how often
    // pulses are sent and what percentage of the time is
    // spent transmitting.
double PRF;                //Pulse Repetition Frequency
double PRI;                //Pulse Repetition Interval
double DutyCycle;          //Percentage spent transmitting
double CPI;                //Number of pulses collected
                           // by GMTI for one data cube
double CPIDuration;        //Time between receipt of scans
                           // for the tracker (minimum is
                           // CPI * PRI)
double ScenarioTotalTime;  //Time entire scenario takes
double DeAlias;            //used to de-alias Doppler
                           // measurements
double ExpSNR;             //Expected SNR for targets, for
                           // use in calculating variances
                           // theoretically
double PerfectRVar;        //variance used for range in
                           // truth data
double PerfectAzVar;       //variance used for azimuth in
                           // truth data
double PerfectDopVar;      //variance used for Doppler in
                           // truth data
  // Note that the below variances may be set either as
  // theoretically derived or statistically derived
  // variances.  Statistically derived variances are used
  // for the delivered implementation.
double RVar;               //variance on range measurement
double AzVar;              //variance on azimuth
                           // measurement
double DopVar;             //variance on doppler
                           // measurement
    //the following three parameters are used to test if
    // the corresponding hypothesis is correct.
double CVHLimit;           //Test limit for Constant
                           // Velocity Hypothesis
double LATHLimit;          //Test limit for Linear,
                           // Accelerating Track
                           // Hypothesis
double ArcHLimit;          //Test limit for Arc of Circle
                           // Hypothesis
    //the following three parameters are used for the
    // values of q in computing the process noise
    // covariance matrix Q.
double cvsq;                 //mean square acceleration
                             // plant noise for constant
                             // velocity hypothesis
double latsq;                //mean square acceleration
                             // plant noise for linear,
                             // accelerating track
                             // hypothesis


double mnsq;                 //mean square acceleration
                             // plant noise for arc and
                             // unmodeled hypothesis
    //the following three parameters are used to decide
    // how long a track may be kept without an update.
double NewLimit;             //Time limit for New tracks
double NoviceLimit;          //Time limit for Novice tracks
double EstablishedLimit;     //Time limit for Established
                             // tracks
double PlatformHeight;       //Height, in meters, of
                             // stationary radar platform
double PlatformXPos;         //X position of radar by the XY
                             // grid used with targets (Y
                             // position is 0)
    //the following six parameters are used for the
    // simple geographic grid structure being used in the
    // Matlab code.  Others would be used for different
    // geographic schemes.
double GridEltXSize;         //X width of one element of the
                             // grid
double GridEltYSize;         //Y width of one element of the
                             // grid
double GridXWidth;           //X width of entire grid (must
                             // accommodate radar scan)
double GridYWidth;           //Y width of entire grid (must
                             // accommodate radar scan)
double GridXNearRatio;       //Ratio to determine nearness
                             // in X
```

```
  double GridYNearRatio;    //Ratio to determine nearness
                            // in Y
  double MinTimeExtent;     //Minimum time within PRI
                            // before GMTI may receive.
  double RangeMultiplier;   //Used in determining range for
                            // range bin to absolute range
                            // calculation.
}
```

3.5 EXTRAPOLATED TRACK POSITIONS

This data structure is a simple container for track extrapolations. A vector could be used instead, if desired.

```
struct ExtrapolatedTrackPosition
{
  double x;                //x coordinate position
  double y;                //y coordinate position
  double xdot;             //x direction velocity
  double ydot;             //y direction velocity
  double Doppler;          //radial direction velocity
}
```

3.6 MUNKRES RESULT

This is a simple vector of integer tuples (or a two column/row matrix of integers) giving back associated track/report pairs as each element. The length of this vector will fluctuate not only with the number of tracks and reports in a given scan, but also with the results of the Munkres algorithm. If all new reports are geographically too far away from all existing tracks to be possibly the same, then the length of the vector would be the Num_Reports + Num_Tracks, whereas the case where all reports and tracks might possibly be associated would result in max(Num_Reports, Num_Tracks) tuples.

```
struct tuple                  //struct for readability
{
  int pair[2];                //holds a track/report
                              //association
}
struct tuple *MunkresResult; //dynamically allocate the
                              // number of these tuples
```

4. IMPLEMENTATION CONSIDERATIONS

The major portion of implementation considerations examined by this report is related to the process of reducing the number of track/report pairs using geographic information. This particular area is focused on because its results determine the difficulty of the rest of the computation (save for Kalman filtering).

The simple loop method listed in Section 2.1.1 for geographic reduction is obviously an expensive way to do geographic reduction and is not preferred. A more desirable method would be to use a custom structure to accept queries. Perhaps the simplest structure which can be used is a grid. A grid would simply be a matrix of track reference lists corresponding to locations on the ground. Every target report would fall within some grid element; the track lists from that grid element and grid elements sufficiently close would then all be returned as the possible tracks. The coarseness of this can be adjusted with the size of grid, the element, and the method for determining which other grid spaces are near enough to the target. More complex database methods could also be used. A more complex idea would be to use two indices; one sorted along x position and the other along y for the track list. A query could then give a position and a distance from that position, and a range select could be performed on both lists and the intersection of the results returned. Other methods might also present themselves from a specific parallel design.

The choice for geographic reduction methods is a careful trade-off. It is done to save computation, but using too complex a method will simply waste computation. Because there is a second round of elimination with the unmodeled test, it does not make sense to choose a method more complicated than performing the unmodeled test on all potential track/report pairs. Where more target detections are expected or the probability of false alarms is high, this method becomes more important. Recall also that extra target reports may enlarge the computational load both present and future, as they will be added as new tracks for the next scan.

Another concern for processing load is frequency of tracks missing some number of target report associations. Tracks will be dropped if enough time goes by with no additional associations, but they are not dropped immediately. The problem which arises is that the last estimated position of the correct track will be significantly farther away from the target report than would be expected for the normal case. The simplest solution for this is to keep the geographic reduction fairly coarse. A more complex solution might be to keep separate structures for tracks depending upon when they were last updated or to include elements of time in the database query capabilities.

The Matlab implementation will use a simple grid method for this computation. The test of nearness will be as follows: if a target is within a certain ratio of the space of the grid element from one border (specified by parameters *GridXNearRatio* and *GridYNearRatio*), then the neighboring element will be considered. If two neighboring elements are being considered, then the diagonal element between them will also be considered. This grid method, with appropriate grid element size and nearness ratios, will be considered coarse enough that tracks which have missed updates will still be within the returned list of possible tracks. This operation will be encapsulated in the function `get_tracks_near()`, so with only the change in this function and the update function `[update_track_locations()]` an alternate method can be used.

Another location which might easily accept optimizations is the `MunkresResult[]` list. It may be decided to not include tracks when associated with null (or reports or even both). This would reduce the immediate amount of data needed to be passed around, but the information about which tracks and reports are unassociated is needed eventually, so it would need to be determined again in some manner.

APPENDIX A

ASPECT ANGLE ESTIMATION

Using the assumption that the radar platform is stationary at coordinates (0,0), aspect angle estimation can be performed in the following manner:

$$t = \begin{bmatrix} x \\ y \end{bmatrix}, v = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}$$

$$\theta = \operatorname{acos}\frac{t \cdot v}{|t| \times |v|} \tag{17}$$

$$r = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} v \tag{18}$$

$$\cos\alpha = \frac{r \cdot t}{|r| \times |t|} \tag{19}$$

If $\cos\alpha - 1$ is smaller than a tolerance limit (10^{-9} used in code), then θ will be returned as the aspect angle. Should this limit fail, then $2\pi - \theta$ will be returned as the aspect angle.

Note that the radar is actually at position (*PlatformXPos*, 0), not the origin, so appropriate translation must be performed on the coordinates.

ACRONYMS

FAT	–	Feature-Aided Tracker
SAT	–	Signature-Aided Tracker
CAT	–	Classification-Aided Tracker
MTI	–	Moving Target Indicator [radar]
GMTI	–	Ground Moving Target Indicator [radar]
PRF	–	Pulse Repetition Frequency
SNR	–	Signal-to-Noise Ratio

CONVENTIONS

The `courier` font is used for programming code/pseudocode and also quotations (with a reduced font size).

REFERENCES

[1] Francois Bourgeois and Jean-Claude Lassalle, "An Extension of the Munkres Algorithm for the Assignment Problem to Rectangular Matrices," *Communications of the ACM*, Vol 14, Dec. 1971, pp. 802-806.

[2] Robert Grover Brown and Patrick Y.C. Hwang, *Introduction to Random Signals and Applied Kalman Filtering*, John Wiley & Sons, Inc., 1997.

[3] James M. Lebak, "Preliminary Design Review: PCA Integrated Radar-Tracker Application," MIT Lincoln Laboratory Project Report PCA-IRT-1, 9 April 2002, issued 6 February 2004.

[4] James Munkres, "Algorithms for the assignment and transportation problems," *J. SIAM* 5, March 1957, pp 32-38.

[5] Duy H. Nguyen, John H. Kay, Bradley J. Orchard, and Robert H. Whiting, "Classification and Tracking of Moving Ground Vehicles," *MIT Lincoln Laboratory Journal*, Volume 13, Number 2, 2002. pp. 275-308.

[6] Roger M. du Plessis, "Poor Man's Explanation of Kalman Filtering or How I Stopped Worrying and Learned to Love Matrix Inversion," Taygeta Scientific Inc., 1996.

[7] Albert I. Reuther, "Preliminary Design Review: GMTI Processing for the PCA Integrated Radar-Tracker Application," MIT Lincoln Laboratory Project Report PCA-IRT-2, 8 April 2002, issued 6 February 2004.

[8] George W. Stimson, *Introduction to Airborne Radar*, second edition. SciTech Publishing, Inc., 1998.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE 25 February 2003	3. REPORT TYPE AND DATES COVERED Project Report

4. TITLE AND SUBTITLE Preliminary Design Review: Kinematic Tracking for the PCA Integrated Radar-Tracker Application	5. FUNDING NUMBERS C — F19628-00-C-0002
6. AUTHOR(S) W.G. Coate	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Lincoln Laboratory, MIT 244 Wood Street Lexington, MA 02420-9108	8. PERFORMING ORGANIZATION REPORT NUMBER PR-PCA-IRT-4
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA/ITO 3701 Fairfax Drive Arlington, VA 22203-1714	10. SPONSORING/MONITORING AGENCY REPORT NUMBER ESC-TR-2003-072
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

A kinematic tracker is a classical tracking system. The idea behind kinematic tracking is to store positions and velocities and then, using this information, associate incoming targets with the stored profiles. Using these new targets, the tracker updates its information about positions and velocities to prepare to do the entire process again when it is given the next set of targets.

14. SUBJECT TERMS			15. NUMBER OF PAGES 42
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Same as Report	19. SECURITY CLASSIFICATION OF ABSTRACT Same as Report	20. LIMITATION OF ABSTRACT Same as Report